SCR algorithm: saving/restoring states of file systems

Full text Pdf (556 KB)

Source **ACM SIGOPS Operating Systems Review** archive
Volume 33 , Issue 1 (January 1999) table of contents
Pages: 26 - 33
Year of Publication: 1999
ISSN:0163-5980

Authors Wei Xiao-Hui
Ju Jiu-Bin

Publisher ACM Press New York, NY, USA

Additional Information: abstract index terms collaborative colleagues peer to peer

Tools and Actions: Discussions Find similar Articles Review this Article
Save this Article to a Binder Display in BibTex Format

DOI Bookmark: Use this link to bookmark this Article: http://doi.acm.org/10.1145/309829.309839
What is a DOI?

↑ ABSTRACT

Fault-tolerance is very important in cluster computing. Many famous cluster-computing systems have implemented fault-tolerance by using checkpoint/restart mechanism. But existent checkpointing algorithms can not restore the states of a file system when roll-backing the running of a program, so there are many restrictions on file accesses in existent fault-tolerance systems. SCR algorithm, an algorithm based on atomic operation and consistent schedule, which can restore the states of file systems, is present in this paper. In SCR algorithm, system calls on file sytems are classified into idempotent operations and non-idempotent operations. A non-idempotent operation modifies a file system's states, and an idempotent operation does not. SCR algorithm dynamically follows the tracks of a program's running, logs each non-idempotent operation used by the program and the information that can restore the operation in disks. When checkpointing roll-backing the program, SCR algorithm will revert the file system states to the last checkpoint time. By using SCR algorithm, users are allowed to use any file operation in their programs.

↑ INDEX TERMS

Primary Classification:
D. Software
↪ **D.4** OPERATING SYSTEMS
↪ **D.4.5** Reliability
↪ **Subjects:** Fault-tolerance

Additional Classification:
D. Software
↪ **D.4** OPERATING SYSTEMS
↪ **D.4.5** Reliability

↪ **Subjects:** Checkpoint/restart

G. Mathematics of Computing
↪ **G.4** MATHEMATICAL SOFTWARE
↪ **Subjects:** Algorithm design and analysis

General Terms:
Algorithms, Design, Performance, Reliability, Theory, Verification

Keywords:
atomic operation, checkpointing, fault-tolerance, recoverability of file systems

Collaborative Colleagues:

Ju Jiu-Bin:	Wei Xiao-Hui
Wei Xiao-Hui:	Ju Jiu-Bin

Peer to Peer - Readers of this Article have also read:

- Data structures for quadtree approximation and compression
  Communications of the ACM 28, 9
  Hanan Samet
- 3D representations for software visualization
  Proceedings of the 2003 ACM symposium on Software visualization
  Andrian Marcus , Louis Feng , Jonathan I. Maletic
- Boundary and Object Detection in Real World Images
  Journal of the ACM (JACM) 23, 4
  Yoram Yakimovsky
- The state of the art in automating usability evaluation of user interfaces
  ACM Computing Surveys (CSUR) 33, 4
  Melody Y. Ivory , Marti A Hearst
- Image-based objects
  Proceedings of the 1999 symposium on Interactive 3D graphics
  Manuel M. Oliveira , Gary Bishop

SCR ALGORITHM: SAVING/RESTORING STATES OF FILE SYSTEMS

WEI Xiao-Hui JU Jiu-Bin
Wei@dcs.jlu.edu.cn jjb@mail.jlu.edu.cn
(Department of Computer Science, Jilin University, Changchun 130023, China)

Abstract Fault-tolerance is very important in cluster computing. Many famous cluster-computing systems have implemented fault-tolerance by using checkpoint/restart mechanism. But existent checkpointing algorithms can not restore the states of a file system when roll-backing the running of a program, so there are many restrictions on file accesses in existent fault-tolerance systems. SCR algorithm, an algorithm based on atomic operation and consistent schedule, which can restore the states of file systems, is present in this paper. In SCR algorithm, system calls on file sytems are classified into idempotent operations and non-idempotent operations. A non-idempotent operation modifies a file system's states, and an idempotent operation does not. SCR algorithm dynamically follows the tracks of a program's running, logs each non-idempotent operation used by the program and the information that can restore the operation in disks. When checkpointing roll-backing the program, SCR algorithm will revert the file system states to the last checkpoint time. By using SCR algorithm, users are allowed to use any file operation in their programs.
Key Words: fault-tolerance , checkpointing, atomic operation, recoverability of file systems.

1 Introduction

Cluster computing systems, such as PVM[1], have been more and more important parallel computing environments due to their good performance/cost and large computing ability. However, the more machines are involved in a computing, the fault rate is higher. Hence, improving the reliability of cluster computing systems is necessary. To provide a reliable computing environment to users, many cluster computing systems realized fault-tolerance by using checkpointing algorithms, such as Condor[2,3], Mist[4,5], CoCheck[6], Fail-safe PVM[7], and Dome[8,9] etc.

A computing is composed of data and program. Checkpointing is usually used to improve the program's reliability. Periodically, checkpointing algorithms take global checkpoint to save the current running states of a program in the stable storage. When a failure occurs, checkpointing algorithms roll-back the program to the last checkpoint cut to mask the failure.

Data are usually saved in files. Usually, two methods are used to improve the availability of a distributed file system [10]. One is to make file systems recoverable, the other is to make file systems robust. A file is recoverable if it is possible to revert it to an earlier, consistent state when an operation on the file fails or is aborted by the client. Recoverable files are realized by atomic updated techniques, and mainly used in data base systems. A file is called robust if it is guaranteed to survive crashes of storage device and decays of the storage medium. Robust files are implemented by redundancy techniques such as mirrored file and RAID[11-14] etc.

In cluster computing systems it is necessary to realize recoverability of file systems for supporting fault-tolerance. File systems are very important environments for programs. During running, a program needs access file systems for reading input or writing output from time to time. So incorrect file system states will lead to program's error running. Existing fault-tolerance systems only rollback a program's running states and dose not restore the file system states correspondingly. If the program executed the operations that modified the file system states, such as delete, rename, ..., the program would rerun in a changed file system environments. Then the program's running is unpredictable. For this reason, Condor, Mist and other fault-tolerance systems disallow their users to access file systems arbitrarily except read-only or write-only operations. Such restrictions prevent general applications from using checkpointing to improve reliability. In [15], random write and read are considered as unrecoverable operations. [4] thinks that write-and-copy technology may be used to resolve the problem, but no further work is reported.

In the paper, SCR algorithm, which is based on atomic operation and consistent schedule, is presented. SCR algorithm develops the concept of file systems recoverability in that it can revert a file system's states changed by a program's running, not only by a single file operation. By using SCR algorithm, synchronous checkpointing algorithms[16] may allow arbitrary file accesses in user applications.

The rest of the paper is organized as the following: Section 2 gives out the important concepts and definitions. SCR algorithm and its proof of correctness are given in section 3. In section 4, an implementation example of SCR algorithm is presented. Section 5 discusses SCR algorithm's performance, and the last section concludes the paper.

2 Concepts and Definitions

Two general strategies can be used to realize saving/restoring the states of a distributed file system corresponding to a distributed program's fault-tolerance running. One is static save/restore strategy (SSR), the other is dynamic save/restore strategy (DSR). In SSR, when checkpointing mechanism checkpointing a program, a copy of the current file system states are also saved in stable storage. When checkpointing mechanism roll backing the program after a failure, the file system states are replaced by the copy. DSR dynamically follows the tracks of a program's running, and logs every such file operation that modifies the file system states and the necessary information that can undo these operations in stable storage. When checkpointing mechanism roll backing the program, DSR revert the file system states to the last checkpoint by undoing all the file operations used by the program.

In general, it is unpredictable how a program will change the file system states during its running. Hence, SSR should save the states of all the file system that are permitted the program to access. Then a lot of unnecessary information is saved, so SSR is much inefficient. Moreover, the high disk space overhead makes SSR difficult to implement. On the contrary, DSR dynamically keeps the tracks of a program's real running, only the useful information is saved. So, DSR is much more efficient and practical than SSR. SCR algorithm is just based on DSR strategy.

In Unix, a file system is composed of files and directories. Applications access file systems by using system calls provided by OS. In SCR, system calls related to file access are classified into idempotent operations and non-idempotent operations. SCR defines a save operation and an undo operation for each non-idempotent operation. A set of stacks, called **undo stacks** in SCR, that work on stable storage (such as disks) are used to save information that can undo the file system changes by a distributed program's running.

Definition 1 A file system's states

A file system's **states** are its directories' and files' name, contents, location, owner, mode, and link number etc. These characters do not rely on any program's running. SCR is responsible for saving and restoring such features of a file system. (Note: A file system's states do not include a file's open mode, open file handle, read and write pointer's offset etc. Because these states can not exist independent of a program's running, these states should be a part of a program's running states. So, they should be saved/restored by the checkpointing mechanism.)

Definition 2 Idempotent operations and Non-idempotent operations

All system calls about file accesses are classified into **idempotent** operations and **non-idempotent** operations. **Non-idempotent** operations are all the system calls that may modify a file system's states. **Idempotent** operations are all the syatems on file systems that never change a file system's states. For examples, in SunOS 4.1.3, create(), mkdir(), remove(), rmdir(), rename(), write(), chown(), chmod(), link(), and unlink() etc are non-idempotent operations; read(), access(), fstat(), stat() etc are idempotent operations. In a broad sense, all idempotent operations are some kind of read operations, and all non-idempotent operations are some kind of write operations.

Definition 3 NOP

Suppose nop_i (i=1, ... ,n) is an non-idempotent operation, op_j (j=1, ... ,m) is an idempotent operation. **NOP** is such a set of file operations that $\forall nop_i$ (i=1, ... ,n) $\in$NOP, and $\forall op_j$ (j=1, ..., m) $\notin$NOP.

Definition 4 Undo Stack

An **undo stack,** composed of a set of **local undo stacks**, works on stable storage such as disks. The data unit of undo stacks is called **undo structure** in SCR. An undo structure logs a non-idempotent operation's name, and the necessary information that can undo the non-idempotent operation. In SCR, every machine holds a local undo stack for a distributed program. When the program executes a non-idempotent operation to a machine's file system, the machine's local undo stack for the program will be pushed into an undo structure at the same time.

Definition 5 Save operations and Undo operations

A **save operation** (sop_i) and an **undo operation** (uop_i) are defined for each nop_i. Sop_is and uop_is are all atomic operations. A sop_i creates and fills the undo structure for the correspondent nop_i and pushes it into the correspondent local undo stack, then finishes the file access of the nopi. An uop_i undoes the nop_i according to the nop_i's undo structure, then pop up the undo structure from the corespondent local undo stack.

3 SCR (Save, Clean, Restore) Algorithm

A distributed program is composed of a set of concurrent processes running on different machines. Consistent schedule is used to coordinate the concurrent file operations of a program. Each machine runs a manage process (MP) to manage the local file systems.

Algorithm Description

Every MP holds a local undo stack for a program, which is NULL at the beginning.

During normal running, when a process of a program is to execute a nop_i to a machine's file system, the process sends a requirement to the machine's MP, and waits for the result. When a process of a program is to execute an op_i, the process operates the file access directly.

At normal time, MPs waits for nop_i requirements in loops. MPs execute the correspondent sop_i and return the result for each nop_i requirement in FIFO order. When executed, a sop_i will push the undo structure for the nop_i into the local undo stack.

When a distributed program taking global checkpoints, every MP cleans its local undo stack of the program.

When a distributed program roll-backed, each MP do uop_is in loops, until all the undo structures are popped up from the local undo stack.

At the end of the program running, MPs clean up all undo stacks of the program.

Proof of Correctness

Suppose a distributed program is composed of **q** concurrent processes running on **p** machines (q>=p), and at the **i** th checkpoint time the file system states are S_i. $S_i = \{S1_i, S2_i, ..., SP_i\}$, Sk_i (k=1,...,P) are the machine k's local file system states. S_0 are the file system states at the beginning of the program running. According to SCR algorithm, at the **i** th checkpoint time all the local undo stacks of the program are null and after the **i** th checkpoint:

1. During normal running, when a process of the program executes a nop_i, the nop_i's undo structure will be pushed into the very machine's local undo stack. In another words, when a machine's file system's states are to be changed, the necessary information that can undo the changes will be saved into the machine's local undo stack at first.
2. When a failure occurs, suppose the global file system states are $Sm = \{S1_{m1}, ..., SP_{mp}\}$. Without losing generality, we also suppose at this time the machine k's(k=1,...,P) local undo stack are {ud_info[1], ..., ud_info[mk-1], ud_info[mk]}. From 1, we know the program has executed mk nops(nop_{i_1}, ..., nop_{i_mk}) since the last checkpoint time(the **i** th checkpoint). Because idempotent operations never vary a file system states, and all file operations in SCR are atomic operations, so the

vary procedure of machine k's file system must be $Sk_{i\_0} \rightarrow Sk_{i\_1} \rightarrow \ldots \rightarrow Sk_{i\_mk}$ (note: Sk_{i_0} is Sk_i, and Sk_{i_mk} is Sk_{mk}). $Sk_{i\_(j-1)} \rightarrow Sk_{i\_j}$ (j=1,...,mk) is caused by nop_{i_j}(j=1,...,mk).

3. When the program roll-backed, machine k's MP will executes mk rops: rop_{i_mk}, ...,rop_{i_1}. And machine k's file system states will take the following changes, $Sk_{i\_mk} \rightarrow$, ... , $\rightarrow Sk_{i_1} \rightarrow$Ski. So the global file system states are reverted to Si, which are just the file system states at the last checkpoint time. At the same time all undo stacks of the program are cleaned up.
4. When it is time to take a new checkpoint, the global file system states are S_{i+1}. At this time all undo stacks of the program are cleaned, then when the next failure occurs, the global file system states will be reverted to S_{i+1} by SCR.
5. At the end of the program running, no information need saving. So, SCR clears all of the program's undo stacks.

Above all, SCR algorithm can revert the file system states related to a program to the last checkpoint time when the program is roll-backed. By using SCR algorithm, user applications can use any file operation in a fault-tolerance system using synchronous checkpointing algorithms.

4 An implementation example of SCR

In the section, we present an implementation of SCR algorithm, which is realized on DPVM[17,18]. DPVM is an enhanced PVM on SunOS 4.1.3, which realized process migration and fault-tolerance by using synchronous checkpointing algorithm.

1 Define undo stack

All local undo stacks work on disks.

(1) Undo Structure

```
Structure undo_info{
        int opflag;        /*      opflag = 1: write()      operation;
                                            2: delete()     operation;
                                            3: rename()     operation;
                                            ...
                           */
        char init_fn[256];         /* file name string */
        char effect_fn[256];       /* file name string */
        char temp_fn[256];         /* file name string */
}
```

(2) PUSH operation

Push an undo_info structure on the top of the local undo stack.

(3) POP operation

Pop the top undo_info structure from the local undo stack. If the undo_info structure's temp_fn item is not null, delete the file whose name is temp_fn.

2. System calls of SunOS4.1.4 about file operations are divided into idempotent operations and non-idempotent operations. In the section, we only use write(), delete() and rename() system calls as non-idempotent operation examples.

2.1 Define NOP

NOP= {write(), delete(), rename()}.

2.2 Define the save and undo operation for write() system calls

(1) save operation – sop1

A file is divided into pages of the same size, and every page has a flag to point out whether the page has been changed from the last checkpoint. Sop1 first determine whether all of the pages are to be written have marked with changed_flag. (a) If there are some pages without being set with changed_flag, sop1 creates a new undo_info structure and a new temp file. Sop1 saves such pages of the file into the temp file, and set these pages with changed_flag at the same time. Then sop1 set the undo_info structure's opflag with 1, init_fn with the name of the file is to be written into, temp_fn with the name of the temp file. After that, sop1 executes PUSH operation and write() operation. (b) If no such pages, sop1 simply executes write() operation.

(2) undo operation – uop1

Uop1 writes the pages saved in the temp file whose name is temp_fn back to the file whose name is init_fn, clear these pages' changed_flag, and executes the POP operation.

2.3 Define the save and undo operation for delete() system calls

(1) save operation – sop2

Sop2 creates a new undo_info structure and generates a temp file name. Then sop2 set the undo_info structure's opflag with 2, init_fn with the name of the file is to be deleted, temp_fn with the new generated temp file name. After that, sop2 executes the rename operation: rename init_fn temp_fn, and executes the PUSH operation.

(2) undo operation – uop2

Uop2 executes the rename operation: rename temp_fn init_fn, and the POP operation.

2.4 Define the save and undo operation for rename() system call

(1) save operation – sop3

Sop3 creates a new undo_info structure, and set the structure's opflag with 3, init_fn with old name of the file, effect_fn with the new name of the file. Then sop3 executes rename() operation and PUSH operation.

(2) undo operation – uop3

Uop3 executes the rename operation: rename effect_fn init_fn, and the POP operation.

3. Add a wrap to each system call(f_syscall()) on file systems:

```
if (f_syscall()∈NOP) then{
        send the f_syscall() requirment to the correspondent MP;
        wait for the result;
}else{
        call f_syscall();
}
```

4. Design the working flow of process MP

(1) During the normal running time

```
While(1){
        Wait for f_syscall() requirments;
        Execute the save operation for f_syscall();
        Return the results to the process who sent the f_syscall() requirment;
}
```

(2) At the time of a program is roll-backed

```
While (the local undo stack is not NULL){
        Execute the undo operation according to the top undo_info structure of the local undo stack;
}
clear the changed_flag of every page of all changed local files;
```

(3) At a new checkpoint time or at the end of a program running

```
While (the local undo stack is not NULL){
        Executes the POP operation according to the top undo_info structure of the local undo stack;
}
clear the changed_flag of every page of all changed local files;
```

5 Discussions on performance

SCR algorithm's overheads lie in two aspects. One is caused by consistent schedule, the other is caused by logging and clearing undo structures dynamically. To reduce the overheads of the first respect relies on improving the schedule policy, and it is our further work. And the following discusses two strategies to reducing the overheads of the second respect.

1.Reducing the information logged in undo structures for each non-idempotent operation. For example, a file is 1M bytes and an application will read and write the file during its running. To reducing the information logged in the undo structure for write() system call, the file is divided into pages like sop1(section 4). If the page size is 50k bytes and a write() system call writes 50K bytes into the file, then the write()'s undo structure need logging 100K bytes(2 pages) at most not 1M bytes.

2.Reducing the undo structures need logging in undo stacks. For example, if you have some knowledge about an application's activities, you can exclude some non-idempotent operations from NOP aggregate.

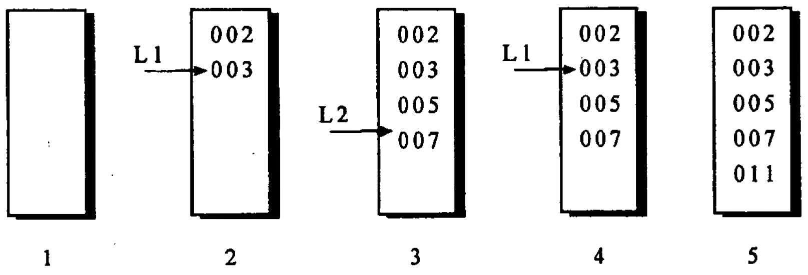

Figure 1 write-only operations

Figure 1 is a program that generates prime numbers. The program writes 002, 003, ..., into the output file during its running. Because the program only executes write operations on the file, so we can exclude write() system call from NOP. Then during the program's fault-tolerance running, nothing will be logged in its undo stacks. However, this doesn't effect the fault-tolerance function. Refer to figure 1, at the begin of the program's running, the output file is NULL(figure1(1)). At the checkpoint time, the output file write pointer's offset is L1(figure1(2)). And when a failure occurs, the file write pointer's offset is L2(figure1(3)). When the program is roll-backed, the write pointer's offset is set to L1, which is the value at the last checkpoint time. However, because nothing is logged in the program's undo stacks, so the output file doesn't revert to the last checkpoint time(figure1(4)). Because the program only executes write() operation to the file, the program still can finish correctly(figure1(5)).

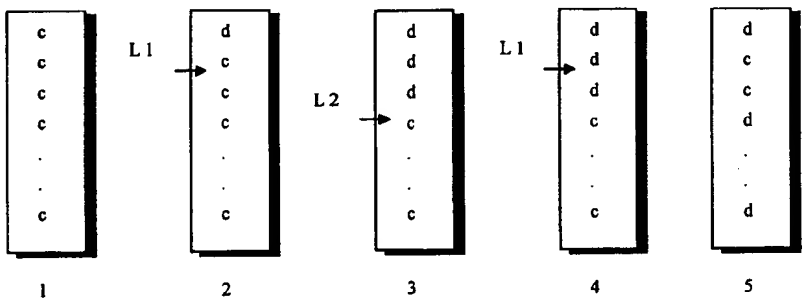

Figure 2 read and write operations

However, for some application NOP must include write() system call, otherwise the program's running is unpredictable. Refer to figure 2, a file's contents are a list of char 'c'(figure2(1)); the program executes the following operations on the file: change all char 'c' to 'd', change all char 'd' to 'c'. At the checkpoint time, the file write pointer's offset is L1(figure2(2)). And when a failure occurs, the pointer's offset is L2(figure2(3)). As the program is roll-backed, the write pointer is set to L1. If NOP does not include write() system call, the file's contents will not be reverted to the last checkpoint time(figure2(4)). Then, at the end of the program's running the file content will be figure 2(5). Clearly, it is not the expected result, which should be a list of char 'd'.The above two examples show when NOP must include write() system call and when needn't. In Condor system etc, users are allowed to use read-only or write-only operations to files, but random write and read operations are inhibited. The reason is also similar with the above description.

6 Conclusion

File systems are very important environments for programs. During running, a program needs access file systems for reading input and writing output from time to time. Existing fault-tolerance systems only rollback a program's running states and dose not restore the file system states correspondingly. For this reason, Condor, Mist and other fault-tolerance systems disallow their users access file systems arbitrarily except read-only or write-only operations. SCR algorithm, based on atomic operation and consistent schedule, can resolve this problem. By using SCR algorithm, users are allowed to use any file operation in their programs. But SCR algorithm only supports synchronous checkpointing algorithms now, how to support asynchronous checkpointing algorithms is our next work in the future.

Bibliography

1. Sunderam V.S. Pvm: a framework for parallel distributed computing. Concurrency: Practice and Experience, 1990,2(4): 315-339.
2. Litzkow M, Marvin. Supporting checkpointing and process migration outside the Unix kernel. In: Proc USENIX-Winter'92, San Francisco, CA, 1992, 283-290.
3. Litzkow Michael J, Miron L, Mattw M. Condor — a hunter of idle workstations. In: IEEE 8ICDCS, San Jose, California, 1988, 104-111.
4. Casas J et al. Mist: pvm with transparent migration and checkpointing. In: Proc 3rd Annual PVM User's Group Meeting, Pittsburgh, 1995.
5. Casas J et al. MPVM: a migration transparent version of pvm. Dept. of Computer Science and Engineering, Oregon Graduate Institute of Science&Technology: Tech Rep CSE-95-002, Feb 1995.
6. Stellner G. Resource management and checkpointing for PVM. In: Proc the 2rd European User's Group Meeting, Lyon, France, 1995, 131-136.
7. Juan León ,Allan L. Fisher, Peter Steenkiste. Fail-safe PVM: a portable package for distributed programming with transparent recovery. School of Computer Science, Carnegie Mellon University: Tech Rep CMU-CS-93-124, 1993.
8. Arabe J N, et al. Dome: parallel programming in a heterogeneous multi-user environment. School of Computer Science, Carnigie Mellon University: Tech Rep CMU-CS-95-137, 1995.
9. Erik Seligmon, Adam Beguelin. High-level fault tolerance in distributed programs". School of Computer Science, Carnegie Mellon University: CMU-CS-94-223, Dec. 1994.
10. Eliezer Levy, and Abraham Silberschatz. Distributed file systems: concept and examples. ACM Computing Surveys, 1990, 22(4): 321-374.
11. P.M.chen, et al. RAID: high-performance, reliable secondary storage. ACM Computing Surveys, 1994, 26(2): 145-185.

12. James S.Plank. Improving the performance of coordinated checkpointers on networks of workstations using RAID techniques. In SRDS-15:15th Symposium on Reliable Distributed Systems, Niagra-on-the-Lake, Canada, Oct.1996, 76-85.
13. J. S. Plank. A tutorial on reed-Solomon coding for fault-tolerance in RAID-like systems. Tenn. University: Tech Rep UT-CS-96-332, July 1996.
14. T.J.E.Schwarz and W.A.Burkhard. RAID: organization and performance. In: Proc. of the 12th Int. Conf. on Dist. Comp. Sys, Yokohama, June 1992, 318-325.
15. James S.Plank. Efficient checkpointing on MIMD architectures[Ph D diss]. Princeton University, Princeton, 1993.
16. D.Manivanan and Mukesh Singhal. A low-overhead recovery technique using quasi-synchronous checkpointing". In: IEEE Proceedings of the 16th ICDCS, Hong Kong, 1996, 100-107.
17. JU Jiu-Bin, WEI Xiao-Hui et al. Implementing process migration in pvm with checkpointing. JOURNAL OF SOFTWARE, 1996, vol 7(3): 175-179.(In Chinese)
18. JU Jiu-Bin, WEI Xiao-Hui et al. DPVM: an enhanced PVM supporting task migration and queuing. CHINESE JOURNAL OF COMPUTERS, 1997, vol 20(10): 872- 877.(In Chinese)